Word Embedding

Ko, Youngjoong
Sungkyunkwan University

Contents
1. Basic Concepts of Neural Network (NN)
2. Why do we need Deep Learning?
3. Learning Representation for NLP
4. Approaches for Word Embedding
   - Ranking-based
   - Word2Vec
   - Glove

Basic Concepts of NN

Perceptron

Multilayer Neural Network
Basic Concepts of NN

- **Multilayer Neural Network (Jeong, 2015)**
  
  The single-hidden layer Multi-Layer Perceptron (MLP)
  
  An MLP can be viewed as a logistic regressor, where the input is first transformed using a learnt non-linear transformation

  $f: \mathbb{R}^D \rightarrow \mathbb{R}^L$
  
  $f(x) = G(h(x) + W^{(0)}w(x) + W^{(1)}x)\ ,\ x $ is the size of input vector $x$
  
  $L$ is the size of output vector $f(x)$

  ![Feed Forward Propagation](image)

- **Training (Weight Optimization)**
  
  $\theta = \{W^{(2)}, b^{(2)}, W^{(1)}, b^{(1)}\}$

  - How to learn the weights?

  "Backpropagation Algorithm"

  
<table>
<thead>
<tr>
<th>최종 결과를 얻고</th>
<th>Feed Forward and Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>그 결과물과 수치가 동일한 결과를</td>
<td>Cost Function</td>
</tr>
<tr>
<td>그 자료를 추정해 나가는 지</td>
<td>Differentiation (이분)</td>
</tr>
<tr>
<td>맵으로 내려가면서 추정하고</td>
<td>Back Propagation</td>
</tr>
<tr>
<td>새로운 Parameter 값을 계산</td>
<td>Weight Update</td>
</tr>
</tbody>
</table>

- **Training (Activation Functions)**
  
  $\text{sigmoid}(a) = \frac{1}{1 + e^{-a}} \quad \text{also called 'logistic function', 'Fermi function'}$

  ![Activation Functions](image)

  $f(x) = \frac{1}{1 + e^{-x}}$

  $\frac{d}{dx}f(x) = f(x)(1 - f(x))$

  $1 - f(x) = f(-x)$

  $2f(x) = 1 + \text{tanh}(\frac{x}{2})$

  Always positive

  famh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$

  ![Activation Functions](image)

  $ f(x) = \frac{\sinh x}{\cosh x}$

  $ e^x = \cosh x + \sinh x$

  $ e^{-x} = \cosh x - \sinh x$

  Output $\in [-1, 1]$

  Faster Backpropagation
Basic Concepts of NN

- **Training (Activation Functions)**
  - Rectified Linear Unit: $f(x) = \max(0, x)$
  - Smooth approximation: "Sigmoid" function
    - $f(x) = \log(1 + e^x)$
    - $f'(x) = e^x / (1 + e^x)$

- **Scoring Functions (Softmax)**
  - $\text{softmax}_j(x) = \frac{e^{x_j}}{\sum e^{x_k}}$
  - $P(Y = i|x, W, b) = \text{softmax}_i(Wx + b)$

Why? Deep Learning

- **Why was not old NN successful? (Jeong, 2015)**
  - Pre-Training: Performance

Why? Deep Learning

- **Pre-Training**
  - Pre-training: $\text{NN}$ 성능이 비약적으로 향상됨
  - AutoEncoder 계열과 Restricted Boltzmann Machine 계열이 있음

Why? Deep Learning

- **Pre-Training-Performance**
  - Regularization hypothesis:
    - Representations good for $P(x)$ are good for $P(y|x)$
  - Optimization hypothesis:
    - Unsupervised initializations start near better local minimum of supervised training error
    - Minima otherwise not achievable by random initialization

[Image references for diagrams and text components]
**Why? Deep Learning**

- **Auto Encoder**

![Auto Encoder Diagram](image)

- The vast majority of rule-based and statistical NLP work regards words as atomic symbols.
  - Walk, natural, language, process

- In vector space terms, this is a vector with one (1) and a lot of zeroes (0).
  - \([0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]\)

- Dimensionality:
  - 20K (speech) – 50K (PTB) – 500K (big vocab) – 3M (Google 1T)

- “One-hot” representation
  - It is a localist representation

---

**Learning Word Representation for NLP**

- For web search,
  - If user searches for “Seoul motel,” we would like to match documents containing “Seoul hotel.”

- But
  - Inner product of motel \([0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0]\) and
    hotel \([0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]\) = 0
  - Our query and document vectors are orthogonal
  - No natural notion of similarity in a set of one-hot vectors

- Could deal with similarity
  - Explore a direct approach where vectors encode it
Learning Word Representation for NLP

- **Continuous representation**
  - Latent Semantic Analysis, Random projection
  - Latent Dirichlet Allocation, HMM clustering
  - Distributed Representation (Neural word embedding)
    - Dense vector
    - By adding supervision from other tasks -> improve the representation
    - Get a lot of value by representing a word by means of its neighbors
    - It's one of the most successful ideas of modern statistical NLP

-government debt problems turning into banking crises as has happened in saying that Europe needs unified banking regulation to replace the hodgepodge

Learning Word Representation for NLP

- **Distributed Representation**
  - Distributed Representation (Jeong, 2015)
    - DNN's pioneer AI methods like DNN have big meaning in the real world because they are not relying on symbols like existing methods.

  ![Representation](image)

  Cat
  One-Hot Representation
  [0, 0, 0, 1, 0, ...]
  Distributed Representation
  [34.2, 93.2, 45.3, ...]

Learning Word Representation for NLP

- **Distributed Representation**
  -유사한 것은 '유사하게' 표현되어야 함
  -Curse of Dimensionality 극복 가능

Apple = 001
Pear = 010
Ball = 100

Distance(Apple - Pear) = Distance(Apple - Ball)

Learning Word Representation for NLP

<table>
<thead>
<tr>
<th>Local Representation</th>
<th>Distributed Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only one neuron (or very few) is active</td>
<td>many features, each of which can separately each be active or inactive</td>
</tr>
<tr>
<td>Cat</td>
<td>Cat</td>
</tr>
<tr>
<td>0, 0, 0, 0, 0, 1, 0, 0, 0</td>
<td>-2.3, 1.0, 4.2, 5.3, 2.3</td>
</tr>
</tbody>
</table>

- One-Hot Representation
- Very Sparse
- Very high dimensionality

Ex) word hash to DB Access?
It means 'Integer' space.

- Word embedding
- Real value space
- Dense
- Low Dimensionality
Approaches for Word Embedding

- Basic idea of learning neural network word embeddings:
  - Define a model that aims to predict between a center word $w_c$ and context words in terms of word vectors
  - A loss (or cost) function, e.g.,
    $$ J = 1 - p(\text{emb}_c | w_t) $$
  - Look at many positions $t$ in a big language corpus
  - Keep adjusting the vector representations of words to minimize this loss (or cost)

- Two algorithms:
  - Skip-grams (SG)
    - Predict context words given target
  - Continuous Bag of Words (CBOW)
    - Predict target word from bag-of-words context

- Two training methods:
  - Negative sampling

Good One – Word Representation

- We can compare words without any extra knowledge such as word net!
**Approaches for Word Embedding**

**Neural Network Language Model (Lee, 2015)**

- **Idea**
  - A word and its context is a positive training sample
  - A random word in that same context $\rightarrow$ negative training sample
  - Score(positive) > Score(neg.)

- **Training complexity is high**
  - Hidden layer $\rightarrow$ output
  - Softmax in the output layer
  - Negative sampling

**Word2Vec: CBOW, Skip-Gram**

- **Remove the hidden layer $\rightarrow$ Speedup 1000x**
  - Negative sampling
  - Frequent word sampling
  - Multi-thread (no loc)

- **Continuous Bag-of-words (CBOW)**
  - Predicts the current word given the context

- **Skip-gram**
  - Predicts the surrounding words given the current word
  - CBOW + DropOut / DropConnect

**Ranking-based**

**Skip-gram prediction**

$$
\begin{align*}
P(w_{t+2} \mid w_t) \\
&\quad P(w_{t+1} \mid w_t) \\
&\quad P(w_{t+2} \mid w_t)
\end{align*}
$$

... turning into banking crises as ...

output context words $m$ word window

output context words $m$ word window
Approaches for Word Embedding

Details of Word2vec (Manning, 2017)

- For each word \( t = 1 \ldots T \), predict surrounding words in a window of "radius" \( m \) of every word.
- Objective function: Maximize the probability of any context word given the current center word:

\[
J'(\theta) = \prod_{t=1}^{T} \prod_{m=-m}^{m} P(w_{t+m} | w_t; \theta)
\]

Negative Log Likelihood:

\[
J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{m=-m}^{m} \log P(w_{t+m} | w_t)
\]

where \( \theta \) represents all variables we will optimize.

Cross Entropy Loss (Sung, 2017)

- Linear model

\[
x \rightarrow \text{Linear} \rightarrow \hat{y}
\]

<table>
<thead>
<tr>
<th>Hours (x)</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>?</td>
</tr>
</tbody>
</table>

- Logistic Regression: pass/fail (0/1)

\[
x \rightarrow \text{Linear} \rightarrow \hat{y}
\]

Cross Entropy Loss (Sung, 2017)

- The objective function – details
- Terminology: loss function = cost function = objective function
- Usual loss for probability distribution: Cross-entropy loss
- With one-hot \( w_{t+j} \) target, the only term left is the negative log probability of the true class
Details of Word2Vec

- Predict surrounding words in a window of radius \( m \) of every word
- For \( p(w_i | w_j) \) the simplest first formulation is

\[
p(o | c) = \frac{\exp(u^T w_o)}{\sum_{o=1}^{v} \exp(u^T w_o)}
\]

- Where \( o \) is the outside (or output) word index, \( c \) is the center word index, \( v_c \) and \( w_o \) are “center” and “outside” vectors of indices \( c \) and \( o \)
- Softmax using word \( c \) to obtain probability of word \( o \)

To train the model: Compute all vector gradients!

- We often define the set of all parameters in a model in terms of one long vector \( \theta \)
- In our case with \( d \)-dimensional vector and \( V \) many words:

\[
\theta = \begin{bmatrix} v_o \vdots v_c \vdots v_u \vdots \vdots \vdots \end{bmatrix} \in \mathbb{R}^{2dV}
\]

- We then optimize these parameters

Note: Every word has two vector. Makes it simpler.
**Approaches for Word Embedding**

### Simple Example of Word Embedding

- "I like a delicious cake."
- delicious | cake

![Diagram showing word embedding](image)

**Calculating all gradients!**

- We went through gradient for each center vector \( \mathbf{v} \) in a window
- We also need gradients for outside vectors \( \mathbf{u} \)

- Generally, in each window, we will compute updates for all parameters that are being used in that window.
- For example, window size \( m = 1 \), sentence:

  "We like learning a lot"

- First window computes gradients for:
  - Internal vector \( \mathbf{v}_{\text{wm}} \) and external vectors \( \mathbf{u}_{\text{wm}} \) and \( \mathbf{u}_{\text{learn}} \)
Approximations

- The normalization factor is too computationally expensive.
  \[ p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \]

- Hence, you will implement the skip-gram model with negative sampling.

- Main idea: train binary logistic regressions for a true pair (center word and word in its context window) versus a couple of noise pairs (the center word paired with a random word)

The skip-gram model and negative sampling

- From paper: “Distributed Representations of Words and Phrases and their Compositionality” (Mikolov et al. 2013)
- Overall objective function: \[ J(\theta) = \frac{1}{n} \sum_{i=1}^{n} J_i(\theta) \]
  \[ J_i(\theta) = \log \sigma (u_i^T v_c) + \sum_{j \sim P(w)} \left[ \log \sigma (-u_j^T v_c) \right] \]

  - Where \( k \) is the number of negative samples and we use, the sigmoid function! \[ \sigma(x) = \frac{1}{1 + e^{-x}} \]
  - (we’ll become good friends soon)
  - So we maximize the probability of two words co-occurring in first log

Slightly clearer notation:

\[ J_i(\theta) = \log \sigma (u_i^T v_c) + \sum_{j \sim P(w)} \left[ \log \sigma (-u_j^T v_c) \right] \]

- Maximize probability that real outside word appears, minimize prob. that random words appear around center word

- \( P(w)=U(w)^{3/4}/Z \), the unigram distribution \( U(w) \) raised to the 3/4 power (We provide this function in the starter code).

- The power makes less frequent words be sampled more often

Approaches for Word Embedding

Why not capture cooccurrence counts directly? (Manning, 2017)

- 2 options: full document vs. windows
  - Word-document co-occurrence matrix will give general topics (all sports terms will have similar entries) leading to “Latent Semantic Analysis”
  - Instead: Similar to word2vec, use window around each word ----> captures both syntactic (POS) and semantic information
Approaches for Word Embedding

- **Example: Window based co-occurrence matrix**
  - Window length 1 (more common: 5 – 10)
  - Symmetric (irrelevant whether left or right context)
  - Example corpus:
    - I like deep learning.
    - I like NLP.
    - I enjoy flying.

- **Problems with simple co-occurrence vectors**
  - Increase in size with vocabulary
  - Very high dimensional: require a lot of storage
  - Subsequent classification models have sparsity issues
  - Models are less robust

- **Window based co-occurrence matrix**

<table>
<thead>
<tr>
<th>counts</th>
<th>I</th>
<th>like</th>
<th>enjoy</th>
<th>deep</th>
<th>learning</th>
<th>NLP</th>
<th>flying</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>like</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>enjoy</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>deep</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>learning</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>NLP</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>flying</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
**Approaches for Word Embedding**

- Combining the best of both worlds: GloVe

\[ f(\theta) = \frac{1}{2} \sum_{i,j=1}^n f(P_{ij}) (u_i^T v_j - \log P_{ij})^2 \]

- Fast training
- Scalable to huge corpora
- Good performance even with small corpus, and small vectors

---

**Approaches for Word Embedding**

- What to do with the two sets of vectors?
  - We end up with \( U \) and \( V \) from all the vectors \( u \) and \( v \) (in columns)
  - Both capture similar co-occurrence information. It turns out, the best solution is to simply sum them up:

\[ X_{final} = U + V \]

- One of many hyperparameters explored in GloVe: Global Vectors for Word Representation (Pennington et al. (2014))

---

**Approaches for Word Embedding**

- Word2Vec 학습파일 포맷
  - -train
  - 한 문장 별로 한 라인에 문장 자질로 구성

- Tutorial

---

**References**

- Kim, Y. “Convolutional Neural Networks for Sentence Classification,” EMNLP, 2014.
- Kim, S., Simple Pytorch tutorial Zero to All, [https://github.com/hunkim/PyTorchZeroToAll](https://github.com/hunkim/PyTorchZeroToAll), 2017.
Thank you for your attention!

http://nlplab.skku.edu

고 영 중